International Journal of Information Security (2025) 24:212
https://doi.org/10.1007/s10207-025-01125-w

REGULAR CONTRIBUTION

# Progressive attack graph: a technique for scalable and adaptive attack graph generation

**Alessandro Palma**[1] · **Claudio Cicimurri**[1] · **Marco Angelini**[2]

Received: 26 June 2025 / Accepted: 29 August 2025


**Abstract**

In modern computer networks where sophisticated cyber attacks occur daily, a timely cyber risk assessment becomes paramount. Attack Graph (AG) constitutes a highly effective solution for performing cyber risk assessment in the context of multi-step attacks on computer networks. However, its construction is hindered by significant scalability challenges arising from the inherent combinatorial complexity of the process. This sequential methodology results in prolonged delays before analytical capabilities can be leveraged. Moreover, due to the extended time required for AG generation, existing techniques poorly capture the dynamic evolution of network structures, thereby limiting their ability to provide real-time adaptability in response to environmental changes.

To mitigate these problems, this paper rethinks the classic AG analysis through **StatAG**, a novel workflow in which the analyst can query the system anytime, thus enabling real-time analysis before the completion of the AG generation with quantifiable statistical significance. To achieve this goal, we leverage progressive data analysis combined with statistical analysis. Beyond the real-time capabilities enabled by progressive computation, we further speed up the AG generation with two algorithms that accelerate the convergence of the statistical significance of generated AG. The former is about the weighted path sampling to avoid the possible high number of collisions introduced by random walks. The latter proposes an approximated version of the Kolmogorov-Smirnov distance linear in the number of attack paths. While statistical significance enables the progressive AG generation for every analysis, we present **SteerAG** to accelerate the generation by steering it with the analysis query. SteerAG leverages Machine Learning (ML) models, specifically decision trees, to learn vulnerability features from already generated attack paths to derive the steering rules enabling acceleration. To show the capabilities of the proposed workflow, we perform an extensive quantitative validation and present a realistic case study on networks of unprecedented size. It demonstrates the advantages of our approach in terms of scalability and fitting to common attack path analyses.

**Keywords** Attack graph · Attack path analysis · Progressive computation · Progressive data analysis · Statistical significance · Computational steering

## 1 Introduction

In today's digital world, where attackers are evolving smarter and the number of different vulnerabilities is growing, cyber-attacks are becoming more complex to mitigate and with a higher impact on organizations' infrastructure [1]. In this scenario, it is crucial for any organization to promptly assess the potential cyber risks their networks are exposed to. This involves identifying the vulnerabilities in the network's systems and quantifying the risk of their potential exploitation, which generally refers to the impact of one or more exploits and their likelihood to happen [2–4]. Among the possible threat models, Attack Graph (AG) [5, 6] is a graph-based representation of the potential attack paths in a network used to analyze its cyber risk. It is currently the primary threat model used by enterprises for threat analysis and risk management [7]: among the others, Security Information and Event Management systems (e.g., IBM QRadar and LogRhythm) leverage AG for attack detection, network monitoring tools (e.g., Zeek and Suricata) integrate modules that generate AG to analyze malicious traffic, and threat intelligence platforms

✉ Alessandro Palma
palma@diag.uniroma1.it

Marco Angelini
m.angelini@unilink.it

[1] Sapienza University of Rome, Rome, Italy

[2] Link Campus University, Rome, Italy

Published online: 20 September 2025

(e.g., ThreatConnect and OpenCTI) use AG to map out threat actor tactics, techniques, and procedures [8].

Although AGs are very expressive attack models, the above systems must face different problems. The first one is the poorly scalable computation of the attack paths, even for networks of moderate size, as AGs may grow exponentially in the size of network hosts and vulnerabilities [5]. Currently, much effort has been spent to find a trade-off between the accuracy of the model and computation time [9]. Indeed, providing scalable approaches for attack path analysis is still an open problem [6, 10]. A second emerging problem is that they postulate a classic analysis workflow in which the generation process comes first and the analysis comes next, aggravating the poor scalability of the generation process, as it slows down or completely stops the analysis in real scenarios [6]. A final problem concerns the alignment between the current network situation and the related AG model. While the classic workflow permits the attack path analysis on the complete set of attack steps, it also implies that any change in the network needs a re-computation of the whole attack graph to reflect such changes, exacerbating the scalability problem [2].

To mitigate these problems, this paper improves the scalability of AG generation by leveraging the foundational aspects of progressive data analysis [11, 12]. The workflow is made of two core components named **StatAG** and **SteerAG** introduced by Palma and Angelini [13]. StatAG leverages the concept of *statistical significance* to express the degree of trustworthiness of a partial AG, where not all the attack steps are computed. It progressively generates AGs, allowing preventive analyses with quantitative quality-controlled statistically significant results. While it leverages random walks and the Kolmogorov-Smirnov test as foundational aspects of statistical significance, we extend the existing solution [13] with two algorithms to further accelerate their computation to enable real-time analyses. While StatAG supports the analysis of any type over AG, when the analysts provide specific security analysis queries, we improve the accuracy of analysis performed on partial AGs through SteerAG. SteerAG is a *steering approach* which automatically accelerates the AG generation based on the requested attack path analysis. Given a path analysis query from an analyst, it models the queried attack path features, extracts from them the features of each single attack step, and uses them to prioritize the generation of attack paths that answer the analysis query. We evaluate the speed-up of this process through a comprehensive scalability analysis performed on thousands of experiments, including different network scenarios and a real case study. The source code and all materials are available as an open source repository at: https://github.com/XAIber-lab/ProgressiveAttackGraph.

The rest of this manuscript is organized as follows. Section 2 introduces the notation of the AG model and how cyber risk is computed, while Section 3 reports related work. Next, Section 4 describes the overview of the approach, with the details of StatAG in Section 5, the algorithms of its improved performance in Section 6, and SteerAG in Section 7. Finally, Section 8 evaluates the approach and Section 9 concludes the paper.

## 2 Background

In this section, we present the fundamental notions of attack graphs and attack path analysis. In particular, we define their components and introduce the risk model used for the attack path analysis.

### 2.1 Attack graph model

An Attack Graph (AG) is a graph-based representation of the possible paths an attacker can exploit to compromise an Information and Communications Technology (ICT) network. It is based on two inputs [6, 14]: the network *reachability graph*, and the network *vulnerability inventory*. A network reachability graph is a directed graph $R = (H, E)$ where nodes are associated with network hosts $h_1, \cdots, h_n \in H$ and edges represent reachability conditions between hosts (taking into account firewall and routing rules). Thus, an edge $e(h_1, h_2) \in E$ indicates that host $h_1$ can communicate with host $h_2$ through a direct link. A vulnerability inventory reports the list of network hosts with an associated set of vulnerabilities, typically obtained by combining the vulnerability knowledge bases, as the National Vulnerability Database (NVD)[1] by NIST, with vulnerability scanners (e.g., Nessus [15]).

With these inputs, AG modeling represents all the dependencies between hosts and vulnerabilities an attacker can exploit. In this paper, we leverage the Topology Vulnerability Analysis (TVA) attack graph model [16], where the nodes are *security conditions* and represent the attacker access privileges in a specific host, while the edges are *exploit dependencies* and represent the possible movement of the attacker in case of successful vulnerability exploit. More formally:

**Definition 1** (Attack Graph) An Attack Graph $G = (V, E)$ is a directed multi-graph in which $V$ is the set of security condition nodes and $E$ is the set of labeled edges where an edge $e = (v_1, v_2, u) \in E$ indicates that the attacker can move from condition $v_1$ to condition $v_2$ by successfully exploiting vulnerability $u$.

Once the AG model is defined, the next step is the *Attack Graph Generation*, which is the computation of the attack

[1] https://nvd.nist.gov/

paths to analyze the cyber risks of the potential attacks. Attack paths represent sequences of compromised devices and vulnerabilities exploited during the attack. They are used to compute scores or metrics like the likelihood of success of an attack, its impact, or the difficulty of execution [17–19].

**Definition 2** (Attack Path) An Attack Path $AP = \langle v_1, u_{1,2}, v_2, u_{2,3}, \cdots v_n \rangle$ is the ordered sequence of attacker states $v_1, \cdots, v_n$, interleaved by the sequence of vulnerabilities $u_{1,2}, \cdots, u_{n-1,n}$ which exploit allows an attacker to move between consecutive states.

### 2.2 Risk model

When AG generation ends, cyber risks are associated with attack paths to analyze attack exposure or address specific requirements (e.g., damage to the network). They are evaluated according to existing approaches [2], that consider CVSS metrics[2] to estimate the likelihood and impact of an attack path $AP$ and calculate the risk according to its standard definition:

$$risk(AP) = likelihood(AP) \cdot impact(AP). \quad (1)$$

The likelihood is calculated using the CVSS-3.1 exploitability metrics (Attack Vector, Attack Complexity, Privilege Required, and User Interaction). The impact is determined by CVSS-3.1 impact metrics. Likelihood, impact, and risk are in the range [0,1] (more details are available in the work by Gonzalez-Granadillo et al. [2]). According to the chosen risk model, the risks of the different attack paths inform the analysis process and consequent decisions on the mitigation strategies. For this reason, we evaluate AG expressiveness through the distribution of the likelihood and impact of its paths.

**Definition 3** (Vulnerability and Attack Path Features) Let $AP$ be an attack path and let $u_{1,2}, \cdots \in U_{AP}$ be the sequence of its vulnerabilities. We refer to *vulnerability features* as the CVSS metrics of a single vulnerability $u_{i,i+1} \in U_{AP}$. We refer to *attack path features* as the metrics for evaluating the risk of attack path $AP$ (for the defined risk model they are likelihood and impact).

Thus, an *attack path analysis* is the answer to queries on attack path features, e.g., $Q = \{impact \geq 0.9 \wedge likelihood < 0.5\}$ represents the query for less probable ($< 0.5$) but very dangerous ($\geq 0.9$) attacks. More formally:

**Definition 4** (Attack Path Query) An attack path Query $Q$ is a query that specifies a range of values for one or more attack path features, potentially in conjunction or disjunction, to express an information need over AG.

[2] https://www.first.org/cvss/v3.1/specification-document

For example, the query $Q = \{impact \geq 0.9 \wedge likelihood < 0.5\}$ represents the need to look at not probable ($< 0.5$) but very dangerous ($\geq 0.9$) potential attacks. Since generating an AG (i.e., enumerating all its paths) and analyzing attack paths is computationally expensive, Palma and Angelini [13] introduced a scalable workflow in which the analysis can be performed before the completion of AG generation, and the analysis queries can steer the generation. In this paper, we further accelerate the generation by improving the sampling algorithm for the statistical significance of the generated attack paths.

## 3 Related work

Several works leverage Attack Graphs (AGs) for analyzing cyber risks of computer networks [5, 6, 20, 21], which underline their expressiveness and high computational complexity. Our workflow is positioned at the intersection of two main research areas: AG generation and attack path analysis. We organized the related work according to them.

### 3.1 Attack graph generation

One possible solution to address the scalability issue of AGs is to leverage distributed and parallel computing. The distributed approaches, as Kaynar et al. [22] and Sabur et al. [23], partition the network based on its services and vulnerabilities and assign the different partitions among distributed agents. Each agent then computes the attack paths on smaller portions of the AG, which are finally combined. Similarly, Chen et al. [24] use a graph segmentation approach to organize the network services into segments, compute the sub-AG for each segment, and merge them based on the connection among the different segments. The approaches based on parallel computing [25, 26] adapt the serial graph search algorithms to multi-core environments. Equivalent approaches have been proposed by Cook et al. [27], who use a hybrid distributed shared memory programming model, Cao et al. [28], who introduce Spark into the AG generation as a parallel computing platform, and Li et al. [25], who propose an OpenMP-based programming implementation. Our approach is agnostic to distribution or parallelization, as it acts on the AG workflow.

Another possibility is using Artificial Intelligence (AI). For example, Li et al. [29] leverage Deep Learning and node2vec [30] to train a neural network with data from system logs and use it to predict dependencies between network hosts and label the sequence of events as potential attack paths. Besides, Ychao et al. [31] model the path discovery problem as a planning problem on graphs. They translate vulnerabilities into formal actions and use state-of-the-art planners to discover attack paths. Additionally, Mao et al. [32] introduce

a Convolutional Neural Network (CNN) with a Graph-based Fusion Module to learn from the detected attacks and extract the high-risk attack chain. Similarly, Zhao et al. [33] predict potential attack targets by leveraging Graph Neural Network and use causality to eliminate confounding elements from the alert dataset, while Presekal et al. [34] propose a hybrid model of Graph Convolutional Long Short-Term Memory (GC-LSTM) and a deep CNN for time series classification.

While Distributed and AI-based solutions provide a good improvement in the scalability of AG generation, they still require waiting until the generation is completed to perform the first analyses. In dynamic scenarios where network components change rapidly (and so do attack graphs), it is not reasonable to wait for the re-computation of the attack paths each time the environment changes. In contrast, the proposed approach allows for querying Attack Path analyses with quality-controlled and statistically significant results at any time.

### 3.2 Attack path analysis

Most current AG-based systems have tackled the scalability issue in cyber risk analysis by prioritizing specific attack paths based on security needs. Some of them prioritize the analysis considering the graph's topological structure, as Sun et al. [18] who use nodes' degree, and Gonda et al. [35] who map AGs to planning graphs and leverage the nodes' centrality. Other works leverage security metrics. For example, Feng et al. [17] propose a greedy algorithm to assign an evaluation value to each node based on attack complexity and asset priority and start generating the paths from the node with the lowest value to its expansion (i.e., the next nodes whose conditions are satisfied for all vulnerabilities). Nichols et al. [36] adapt AG to cyber-physical systems and assign generic priorities to physical components to prioritize the attack path analysis involving them. Similarly, Wang et al. [37] prioritize attack paths on the degree of matching between attack nodes and a set of predefined security conditions. These works take specific assumptions on priorities and, therefore, are fine only for scenarios that conform to them. Moreover, they are static and do not follow the network exposure.

Few works addressed the problem of considering attack path analysis to prioritize the generation. Yuan et al. [19] and Salayma et al. [38] model the AG through a graph database and perform attack path analysis by suitably writing queries to the database. While these solutions allow the expression of complex customizable queries, they have the drawback that they depend on the graph database, which may slow down the performance during the analysis [39]. Differently, other works prioritize AG generation based on intrusion alerts. The idea behind these works is to correlate similar alerts according to statistical methods [40–42], Bayesian networks [43–45], or semantic analysis [46]. Among them, Nadeem et al. [47] use alerts to drive the generation process. They translate alert events to episode sequences that are then used to build the AG and execute the analysis driven by alerts. Similarly, Hassan et al. [48] perform rule matching on system logs to identify the events that match attack behaviors described in security knowledge bases. Next, they generate a provenance graph from the logs to show the sequence of causally related alerts, capture temporal ordering, and assign ranks to the possible attacks. While these alert-based approaches are context-aware, they may not represent the entire environment, such as network components with no alerts detected. In contrast, the proposed approach assesses the entire network without introducing bias to a specific analysis. On the contrary, it dynamically adapts to different queries.

## 4 Overview of the approach

The classic process of attack path analysis [6], reported in Figure 1-a, comprises two milestones: the AG generation and the analysis of its attack paths. The foundation of this workflow is the representation of all the possible attack steps to provide a comprehensive view of the potential attacker's movements on the network. The problem of enumerating the attack paths is combinatorial in the number of edges and determines one of the primary sources of their poor scalability. Consequently, the attack path analysis may never be performed because of this complexity, even for medium-sized networks, or the analyst should make assumptions in the generation, limiting the expressiveness and validity of the results. Palma and Angelini [13] introduced a new workflow for AG generation and analysis that leverages the fundamental concepts of *progressive data analysis* [11, 12], reported in Figure 1-b. The main idea is to produce partial results during execution with increasing accuracy to generate intermediary outputs, potentially targeting interactive times.

In the case of AG generation and analysis, the workflow considers the computation of *early partial* results that would give a coarse-grained approximation of the complete AG. At a certain instant *t_significant* (driven by the amount of processed data), the partial AG becomes *statistically significant* (StatAG), in the sense that it is a good statistical approximation of the complete AG. Statistical significance must consider the AG structure for its formulation. From that moment on, attack path analysis generates *mature partial* results, which reflect the outcome within an acceptable margin of error, quantifiable through statistical indicators.

As the progressive AG generation continues, there will be a specific instant *t_stable* when the statistical significance reaches a threshold that guarantees *definitive partial* results. In these cases, the attack path analysis produces results that will no longer change substantially from the exact and, as such, support tasks like confirmatory analysis. This final part

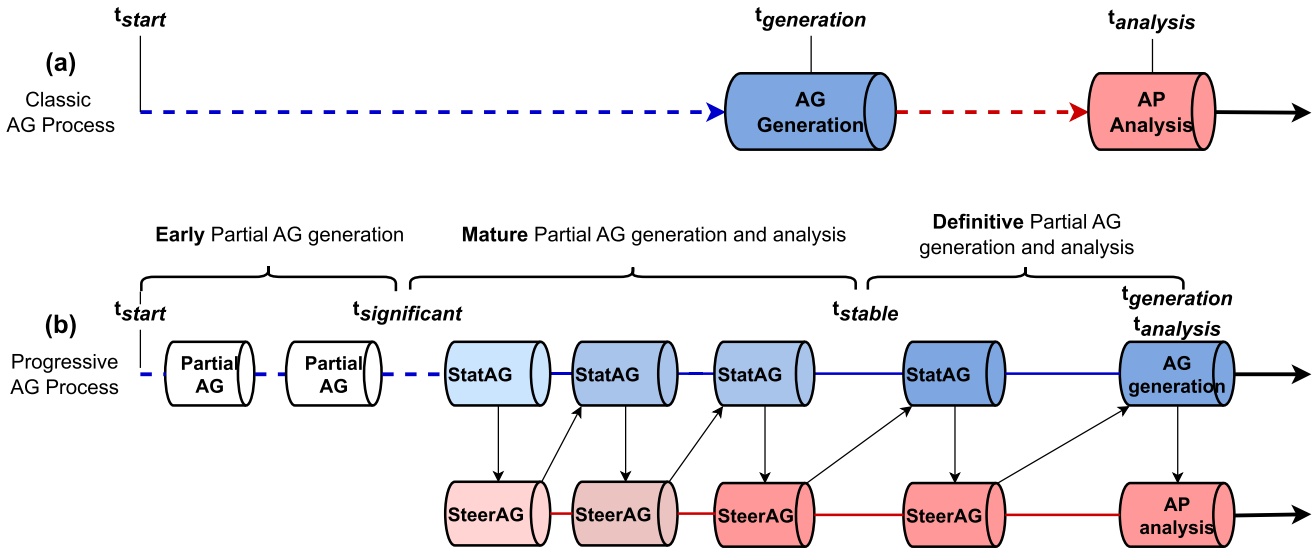



**Fig. 1** (a) Classic attack graph generation and analysis process and (b) the progressive one. Dashed lines indicate periods in which the analysis is in an awaiting state

runs until the complete AG is generated or paths are analyzed. This way of generating AGs allows the analysis of attack paths with preliminary data informed by quantifiable statistical indicators. However, given the combinatorial size of AGs, it may still require a long time to reach the stability of definitive partial results. For this reason, a *steering mechanism* (SteerAG) accelerates the convergence of the AG generation to the final outcome of any analysis executed during the generation [49]. It consists of informing the next generation of the current attack path analysis. For example, if the analysis query asks for the attack paths with the highest risk, then the steering mechanism prioritizes the generation of those high-risk paths.

To automatically steer the generation with the analysis query, Machine Learning (ML) models are used to learn the vulnerability features from the structure of the attack paths and prioritize the generation by selecting the vulnerability according to the learned features. For this reason, the first phase of AG generation must be agnostic to the attack path analysis (white blocks in Figure 1-b) because it is used for collecting the initial balanced small set of attack paths that will be used to train the ML model and label attack paths as answers to the query or not. Consequently, the ML model is trained with the collected labeled attack paths to predict the vulnerability features representative only of attack paths that answer the query positively. This steering mechanism allows a faster convergence of the partial AG to the portion of the complete AG that answers the attack path analysis query.

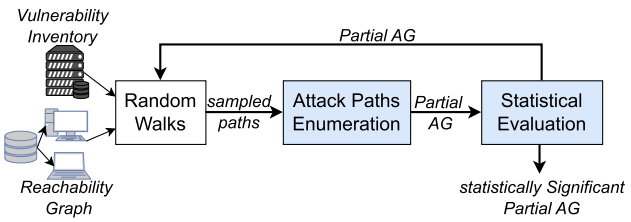



**Fig. 2** StatAG workflow

## 5 StatAG: Statistically Significant Generation

*Statistically Significant Attack Graph Generation* (StatAG) is the first core module of the novel way to generate AGs, and its workflow is reported in Figure 2. It considers partial AGs being aware of statistical significance that becomes progressively higher until it reaches its maximum value corresponding to the complete AG (i.e., the exact result). The most expensive activity during AG generation is the attack path enumeration, and the heavy computational resources required during this task are reflected in the subsequent steps of the workflow. While existing approaches leverage common algorithms for graph traversal, i.e., BFS or DFS [50], we generate attack paths using random walks [51] over the reachability graph. A random walk is a path made of a succession of random steps in the graph. The rationale for random walks is their capability of generating unbiased samples, contrary to the classic traversal algorithms, which depend on the starting node. Thus, they facilitate a quicker convergence of the path features of the partial AG to the complete one [51].

The next step involves constructing the corresponding attack path. As the modeled AG is a multi-graph, multiple edges (i.e., vulnerabilities) may exist between two nodes,

although only one of the multi-edges can be part of an attack path, according to Definition 2. When this happens, the construction of the attack path involves randomly selecting one vulnerability uniformly at random so that the total number of attack paths aligns with the number of sampled walks. The collection of attack paths constructed from the sampled walks defines the partial AG.

**Definition 5** (Partial Attack Graph) Let $G = (V, E)$ be a complete Attack Graph, then a Partial Attack Graph $PAG_i = (V_i, E_i)$ is composed of a subset of nodes $V_i \subset V$ and edges $E_i \subset E$ retrieved after $i$ iterations of random walk sampling.

At this point, the approach iterates toward the generation of other partial AGs that are progressively added to reconstruct the complete AG, eventually. At the end of each iteration, we evaluate the statistical significance of the partial AG that quantifies the degree of its approximation to the complete AG. If it is acceptable, then the partial AG can be used for the initial exploration of the AG, along with indicators communicating its degree of uncertainty. To measure the statistical significance, we need to define the *null hypothesis* $H_0$, which claims that no relationship exists between two sets of data being analyzed [52]. In the context of the proposed approach, the two sets of data are the attack path features of the partial AG and the complete AG, and the statistical significance refers to the probability of rejecting the null hypothesis. To evaluate this probability, we need to measure the probability $p$ of obtaining test results at least as extreme as the observed results (namely *p-value*) and the probability $\alpha$ of rejecting the null hypothesis (namely *significance level*). According to these definitions, the partial AG is statistically significant when $p \leq \alpha$, with the value of $\alpha$ commonly set to 0.05 [52].

We evaluate the null hypothesis through the Kolmogorov-Smirnov (KS) statistical test, which compares two data distributions and quantifies the distance between them [53]. In particular, we evaluate the distance between the attack path feature distributions of the complete and partial AGs. Let $\mathcal{D}_{AG}(x)$ and $\mathcal{D}_{PAG}(x)$ be the distribution of the attack path feature $x$ according to the complete and partial AG, respectively. The KS distance of the two distributions is:

$$KS(x) = sup_x \mid \mathcal{D}_{AG}(x) - \mathcal{D}_{PAG}(x) \mid, \tag{2}$$

where $sup_x$ is the supremum function, which corresponds to the least upper bound of the distances between the two distributions of $x$. According to this definition, we can define the statistical significance of partial AGs.

**Definition 6** (Statistical Significance for partial AG) Let $AG$ be the complete attack graph and $PAG_i$ be the partial attack graph generated at the $i^{th}$ iteration. Let $H_0$: $KS(x) > T$ be the null hypothesis where $KS(x)$ is the KS distance between the distributions of the attack path feature $x$ for the complete and partial attack graph, and $T$ a predefined distance threshold. Then, $PAG_i$ is *statistically significant* for the attack path feature $x$ if the $p \leq \alpha$, with $p$ the p-value of the $KS(x)$ and $\alpha$ the significance level.

This definition is appropriate only for a posteriori evaluation of the statistical significance because it requires the complete attack graph $AG$. To estimate the statistical significance in a real-time application, we introduce the concept of *Attack Path Feature stability* (or simply stability) to quantify the variability of the partial results of an attack path feature gathered during the different iterations. To measure the stability of an attack path feature $x$, we consider the KS distance between the cumulative distribution of $x$ until the $i^{th}$ iteration and the one at iteration $i + 1$. It indicates how much the new samples of iteration $i + 1$ vary from the already sampled distribution, with the rationale that a higher KS distance corresponds to more unstable results, given the higher variability.

**Definition 7** (Attack Path Feature Stability) Let $PAG_i$ be the partial attack graph after $i$ iterations of the approach and $\mathcal{D}_{PAG_i}(x)$ the cumulative distribution of the attack path feature $x$ in $PAG_i$. The stability $\Delta$ of $x$ over $PAG_i$ is:

$$\Delta_{PAG_i}(x) = 1 - \mid \mathcal{D}_{PAG_{i-1}}(x) - \mathcal{D}_{PAG_i}(x) \mid \tag{3}$$

Given that KS distances are defined in the range [0, 1], we subtract 1 from the difference of distances to express the rationale that a higher stability value corresponds to a more significant partial AG.

In the rest of this section, we validate the theoretical formulation of statistical significance and its correlation with the attack path feature stability.

### 5.1 Experimental settings

We validate StatAG with an experimental setup designed for approaching the different factors that affect the AG scalability [6]. It consists of synthetic networks and vulnerability inventories, in which we vary the number of hosts, vulnerabilities, network topology, and composition of vulnerabilities per host. Looking at the former, we considered configurations with up to 20 hosts and 50 vulnerabilities per host, which is the biggest configuration possible to compute all the paths (complete AG, used as ground truth) in a reasonable amount of time (in the order of hundreds of hours) [18]. We then tested 3 types of network topologies, including mesh, random, and power law. We considered 5 levels of heterogeneity of vulnerabilities, varying from 0% (i.e., all vulnerabilities among the hosts are the same) to 100% (i.e., all vulnerabilities among the hosts are different) to increase the variability of vulnerability features and 4 vulnerability distributions (Uniform, Pareto,

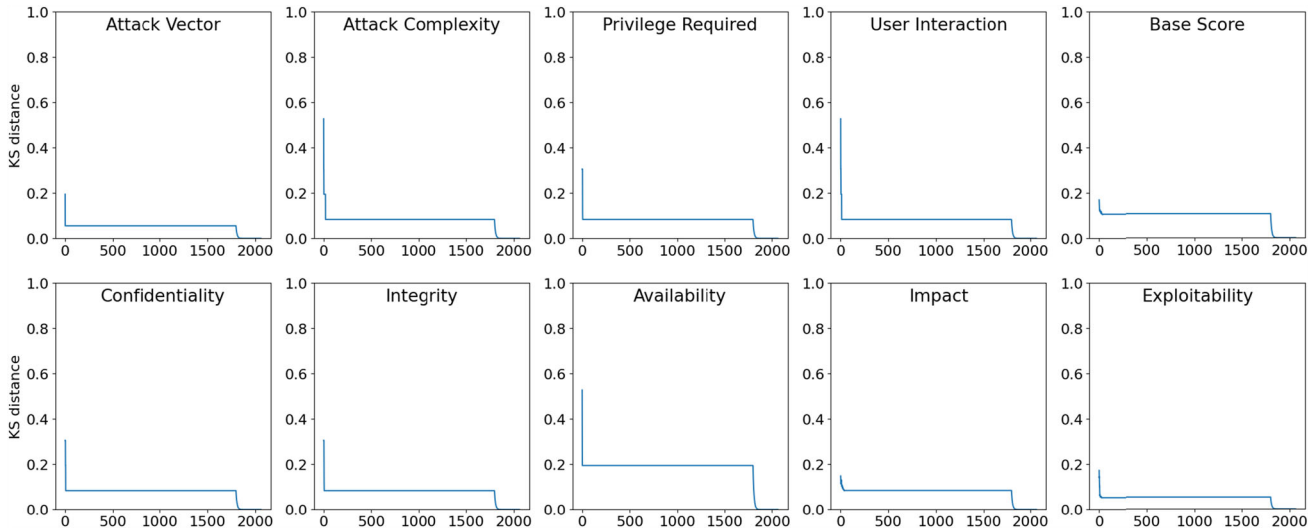



**Fig. 3** KS distance of partial AGs from the GT (vulnerability features)

Binomial, and Poisson) to simulate realistic scenarios. Thus, for each network, we evaluated 60 different configurations, which we varied in 100 statistical variations for a total of 6000 experiments. We use the scikit-learn library [54] for the implementation, and we run the experiments on a Linux server with Intel(R) Xeon(R) Gold 6248 CPU 2.50GHz and 256 GB memory.

## 5.2 StatAG validation

We validate StatAG by studying the convergence of the partial AGs to the complete AG. For the sake of simplicity, in the rest of this section, we refer to the complete AG as *ground truth* (GT) to indicate that it corresponds to all the paths generated at the end of the classic generation process.

The first validation step concerns the convergence of the vulnerability feature distributions (from CVSS) of the partial AGs to the GT. Figure 3 reports the median KS distances during the entire approach execution. The convergence speed of the vulnerability feature distributions depends on the distance threshold $T$ (see Definition 6). For example, considering a good approximation as $T = 0.1$ (corresponding to just 10% approximation of the GT), 9 out of 10 features are statistically significant after just 100 iterations, except for the availability feature that has a KS distance of 0.2 after 100 iterations. In particular, we can notice three different trends in all the plots in Figure 3. The first one coincides with early partial results, where the KS distance from the GT is too high to perform analysis with a reasonable approximation but suitable for initial exploration. This trend lasts about the first 100 iterations. After 100 iterations, corresponding to 9% of the complete execution, the trend is flat until the $1700^{th}$ iteration, corresponding to 77% of the complete execution.

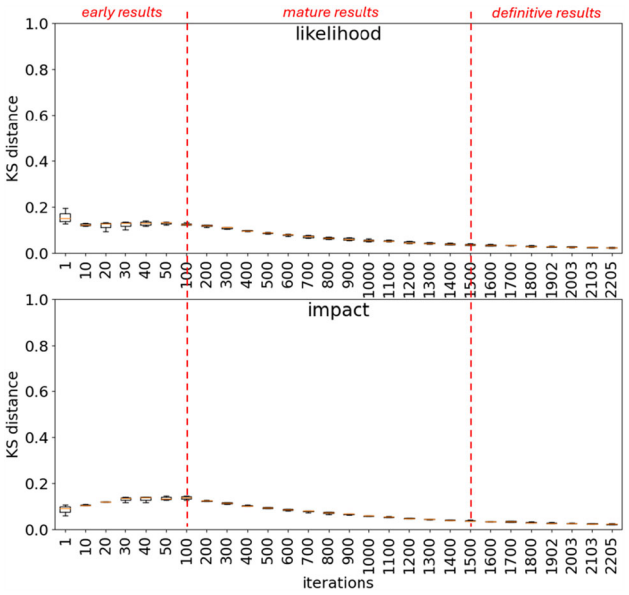



**Fig. 4** KS distance of partial AGs from GT (attack path features)

This is the interval where mature partial results appear, which provide a 10% approximation of the GT for most vulnerability features, with the worst performance reached by the availability with a 20% approximation. Finally, in the last 23% of the execution, the KS distance becomes very close to 0, corresponding to the definitive partial results. Consequently, the vulnerability variability is captured in just 9% of the total iterations (the moment at which mature partial results appear), identifying the iteration from which an analyst can start reasoning on the AG. The convergence trend of the vulnerability features to the GT shows the capability of the progressive approach to represent 90% of the vulnerability inventory after a few iterations. It is a good result for the

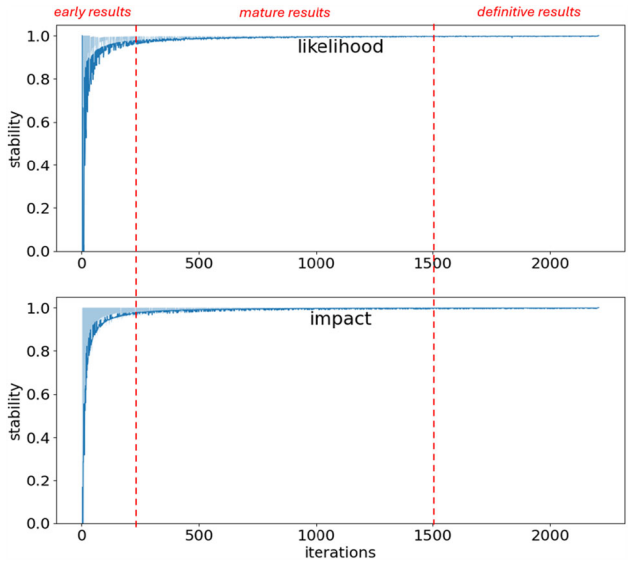



**Fig. 5** Stability for the path features

analysis of the vulnerability inventory, but we need further investigation concerning the attack path analysis.

To this aim, the following validation step concerns the convergence trend of the attack path features of the partial AGs to the GT ones. Figure 4 reports the distribution trend of the KS distances between the partial AGs and the GT for the attack path features, i.e., likelihood and impact. The trends show a gradual convergence for both likelihood and impact, with the largest distance equal to 0.2 in the first 100 iterations, where the variability of the data is also higher (i.e., the box size of the boxplots). The KS distance distribution becomes less variable and has lower median values after about 100 iterations (9% of the execution) and until 1500 iterations (68% of the execution). It is the case of mature partial results, which approximate the attack path features distribution from 15% to 5% accuracy (i.e., KS distances from 0.15 to 0.05). After 1500 iterations and until the end, we can recognize definitive partial results with a KS distance very close to 0. In summary, attack path features can be queried after a few iterations with a 20% approximation, gradually decreasing, reaching a 5% approximation after 1500 iterations, comparable to vulnerability features (Figure 3).

In the final validation step, we furnish evidence demonstrating that the attack path feature stability (Equation 3) accurately reflects the trend in statistical significance so that it can be used as a real-time indicator of the convergence to the GT. To this aim, Figure 5 reports the stability trend for the performed experiments, where the trend of the median values is represented with full-color hue, while the alpha blended areas identify upper and lower quartile values to report the statistical variability.

The stability trend is coherent with the KS distance trend of Figure 4: during the first 100 iterations (9% of the execution), the stability indicates early partial results as well as the KS distance with the GT, with stability values lower than 0.85. Between iterations 100 and 1000 (46% of the execution), the stability reports an intermittent trend between 0.85 and 0.95, indicating the mature partial results, where the AG is assisting its statistical significance. The same trend corresponds to the KS distance from the GT. Finally, we can identify the definitive partial results in the last 54% of iterations, where the stability is very close to 1 and, correspondingly, the KS distance with the GT is very close to 0.

In conclusion, this validation experimentally proves the correlation between stability and statistical significance in different configurations. While statistical significance can be calculated only a posteriori, stability has the great advantage of being computed in real-time, thus informing the analyst at each iteration. The StatAG validation showed the ability of the approach to generate the attack paths progressively, with three thresholds of significance (early, mature, and definitive) that inform about the convergence of the partial AG to the GT. This progressive approach represents an innovation in the Attack Graph community, where traditional solutions typically focus on the complete generation of AGs. It allows continuously fed data from the AG at every iteration and is more efficient in computing a statistically representative version of the AG. However, when it comes to analyzing the complete AG for attack path analysis, this approach still requires enumerating all paths. Since the computation times for the stability analysis, although satisfactory, were not optimal in the previous solution [13], we expedite the calculation of the stability as a real-time metric. In the next section, we contribute a novel, more scalable approach for its computation.

## 6 Improving efficiency of StatAG

While StatAG provides semantically valid results as the convergence to the ground truth, two main challenges remain in terms of performance. First, the number of collisions is very high in the last part of the computation (as shown by the long tails in Figure 4) and, generally, when networks are small. Such a high number of collisions causes waste of computational resources, which is a problem in computationally expensive tasks like AG generation. Second, the stability metric is evaluated according to the KS distance, which has a computational cost of $O(n \cdot log(n))$ [53] with $n$ being the number of paths. This computation is unsustainable when the number of attack paths grows exponentially, potentially losing the real-time capabilities.

For these reasons, this section presents the proposed solutions to make StatAG suitable for real-time environments by improving its scalability. To do so, we first introduce a weighted path sampling algorithm to avoid the high number

of collisions and save resources. Consequently, we propose a revised computation of the KS distance that works in $O(n)$.

## 6.1 Weighted attack path sampling

The main issue related to the high number of collisions during sampling deals with the total randomness of selection, disregarding the information about the probability of a node to be the target node of the attack path (i.e., the last node). For example, if a node $v \in V$ has $|V_1|$ sink neighbors (i.e., they do not allow continuation of the attack paths) and $|V_2|$ standard neighbors (i.e., they allow for longer paths), then the random sampling has probability $\frac{|V_1|}{|V_1+V_2|}$ to stop before reaching the configured length of the random walk. To reduce this probability, we weight the nodes based on the number of attack paths reachable from them. Algorithm 1 reports the pseudocode of the weighted path sampling.

**Algorithm 1** Weighted attack path sampling.

**Require:** $n$ number of nodes.
**Require:** $l$ sampled path length.
1: $A \leftarrow [\varnothing]$
2: $W \leftarrow [1]$
3: $current \leftarrow$ random host
4: **repeat**
5: $candidates \leftarrow$ successors$(current) \setminus A$
6: **if** $candidates = \emptyset$ **then**
7: **return** $A$
8: **end if**
9: $weights \leftarrow [W(current, next) \mid next \in candidates]$
10: $next \leftarrow$ random $c \in candidates$
11: $A \leftarrow A \cup \langle current, next \rangle$
12: $current \leftarrow next$
13: **until** $|A| = l$
14: **for** $i = 1$ to $|A|$ **do**
15: $e \leftarrow A[i]$
16: $W(e) \leftarrow W(e) + weight(|A|, i)$
17: **end for**
18: **if** $K$ iterations passed **then**
19: **for** $e \in W$ **do**
20: **if** $W(e) > T$ **then**
21: $W(e) \leftarrow \lceil W(e)/T \rceil$
22: **end if**
23: **end for**
24: **end if**
25: **return** $A$

The algorithm keeps track of the weights assigned to the different hosts during the generation through the vector $W$, which is a dictionary from edge to attack path count, initialized to 1. It generates a reachability path of length $l$ using some weights to decide the traverse probability of each edge. The algorithm updates the weights based on the edges traversed by the current path (line 5), giving different weights based on the position of the edge in the path (line 9). In particular, edges near the end and the beginning of the path are less weighted because they might correspond to sink nodes. However, if we give more weight to the central nodes, the system will stop generating paths that contain external nodes. For this reason, every time $K$ paths have been generated, the algorithm updates the weights of all edges, lowering the weights of those that have exceeded a certain threshold $T$. This makes sure that even the less frequented edges have a chance of being sampled, as we do not want to introduce a bias towards the central nodes.

### 6.1.1 Weighted attack path sampling evaluation

To evaluate the improvement of the weighted attack path sampling algorithm, we show how the convergence of the KS distance is faster with respect to the simple random walk. We use the setting configurations presented in Section 5.1 and compare the random walk results and the proposed weighted attack path sampling. Figure 6 reports the KS distance from the ground truth for the increasing number of iterations.

Comparing the two trends, we can observe that the distance to the proposed algorithm converges faster in the early stage of the approach. This is further highlighted by longer tails in Figure 6 (right). Additionally, the proposed algorithm reaches smaller KS distance values, thus indicating a slight improvement in the distribution coverage of the attack path features.

To complement this analysis, Figure 7 compares the attack path feature distribution with random walk (Figure 7a) and the weighted attack path sampling algorithm (Figure 7b). The charts display the distribution of the ground truth (i.e., the complete AG), as well as the algorithm's output at the first, middle, and final iterations. The improvement is particularly visible by comparing the *impact* charts. In Figure 7a, the KS distance in the middle iteration is much bigger than the one in Figure 7b, indicating that the convergence process is both quicker and that the distribution converges more similar to the ground truth.

## 6.2 Efficient stability computation

In the progressive workflow for AG generation, the stability metric plays a pivotal role as it drives the analysis of attack path queries. As described in Section 5, it is computed through the Kolmogorov-Smirnov distance [53]. The standard algorithm to compute the Kolmogorov-Smirnov distance is $O(n \cdot \log n)$ where $n$ is the number of paths in the larger distribution. However, such a computational cost is unsustainable when the number of attack paths grows exponentially, making the system unusable.

To overcome this issue, we designed a specialized version of the Kolmogorov-Smirnov distance that is linear in the number of paths generated in the current iteration. This comes at the price of slightly losing accuracy, although without significantly impacting the analysis.

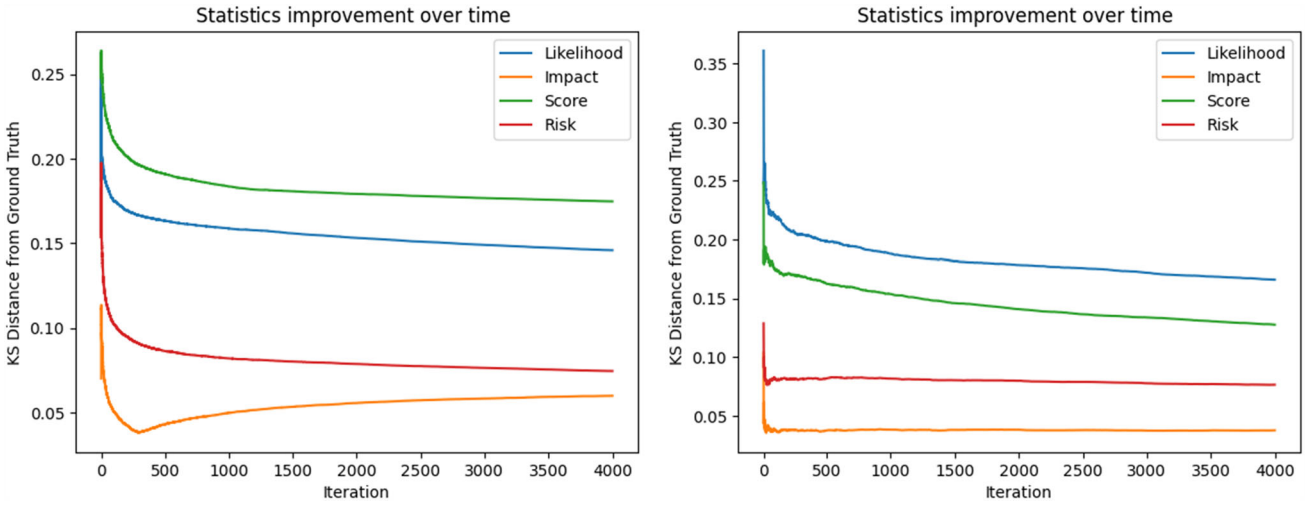
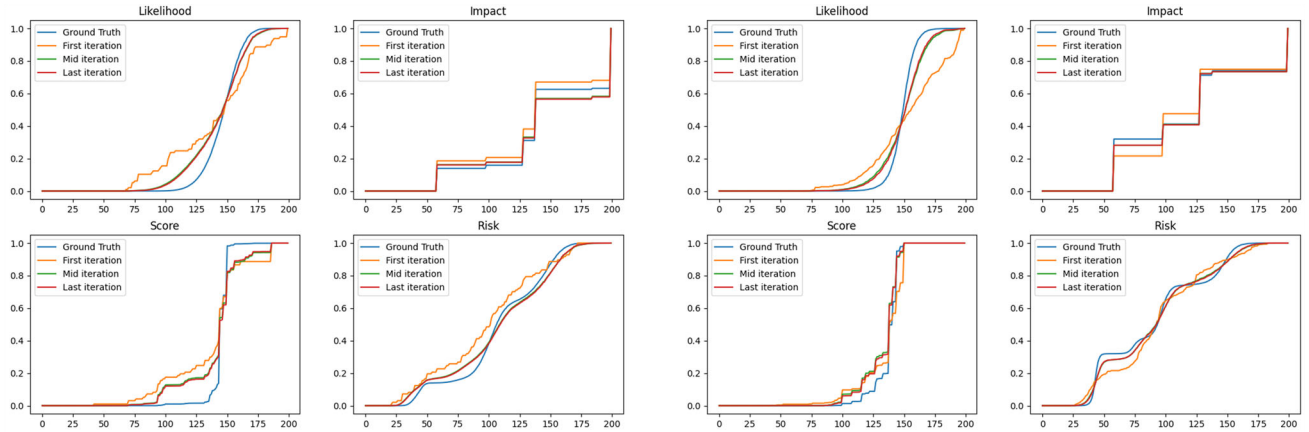



**Fig. 6** KS distance convergence comparison between random walk (left) and weighted path sampling (right)




(a) Attack path feature distribution with random walks.

(b) Attack path feature distribution with weighted path sampling.

**Fig. 7** Comparison of attack path features with (a) random walk and (b) weighted path sampling

The Kolmogorov-Smirnov algorithm performs the following operations [53]:

1. Sorts the two lists of samples

   $X = [x_1, x_2, \ldots, x_n]$ where $x_i \leq x_j$ for $i < j$

   and equivalently for $Y$;
2. Converts each to a cumulative distribution:

$$\hat{F}(x) = \frac{1}{n} \sum_{i=1}^{n} I(x_i \leq x)$$

3. Computes the maximum distance between the two cumulative distributions:

$$\mathrm{KS}(X, Y) = \max_{x \in X \cup Y} |\hat{F}(x) - \hat{G}(x)|$$

To speed up this process, we can leverage the advantages of the attack path metric distributions and how KS distance is used to compute stability.. In fact, we know in advance that the range of the distribution is [0, 10] from CVSS. Additionally, we know that the two sample sets are not random, but the second one contains the first one. This motivates using a bucket sort algorithm, where we clone the distribution for the first sample and update it with the metrics from the paths generated at the i-th iteration to produce the second distribution. The detailed approach is reported in Algorithm 2.

Let us note that the algorithm is equivalent to the standard Kolmogorov-Smirnov distance on the two distributions after applying the same rounding inherently applied by the bucket sort algorithm. However, its cost is $O(n)$, with $n$ being the number of attack paths.

**Algorithm 2** Scalable stability

**Require:** $M$ tally of attack paths.
**Require:** $f$ attack path features.
**procedure** ScalableStability($f$)
  $b \leftarrow$ number of buckets in $M$.
  $C_1 \leftarrow$ To-CDF($M$)
  **for** $x \in f$ **do**
    $i \leftarrow \lfloor \frac{x}{b} \rfloor + 1$
    $M[i] \leftarrow M[i] + 1$
  **end for**
  $C_2 \leftarrow$ To-CDF($M$)
  $KS \leftarrow 0$
  **for** $i = 1$ to $b$ **do**
    $KS \leftarrow \max(KS, |C_1[i] - C_2[i]|)$
  **end for**
  **return** 1 - $KS$
**end procedure**
**procedure** To- CDF()$M$)
  $sum \leftarrow 0$
  $CDF \leftarrow []$
  **for** $i = 1$ to $b$ **do**
    $sum \leftarrow sum + M[i]$
    $CDF[i] \leftarrow sum$
  **end for**
  **for** $i = 1$ to $b$ **do**
    $CDF[i] \leftarrow \frac{CDF[i]}{sum}$
  **end for**
  **return** $CDF$
**end procedure**

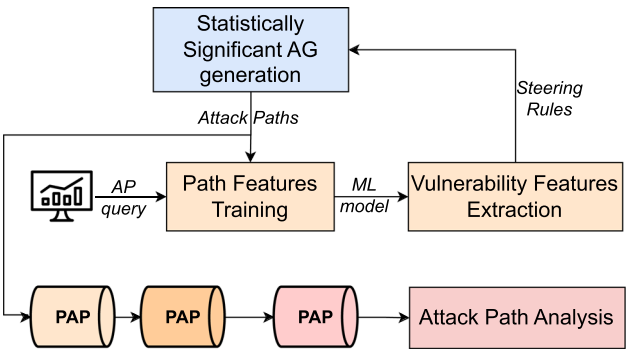



**Fig. 8** Comparison of scalable stability with the standard KS distance algorithm

#### 6.2.1 Scalable stability evaluation

To assess the advantages of the proposed algorithms, we investigate the experimental time and the absolute error for the increasing number of iterations of the approach. We compare the standard KS distance algorithm with the proposed approach for different bucket sizes (10, 100, and 1000 bins). Figure 8 reports the results of the time and absolute error with respect to the state-of-the-art algorithm for KS distance.

The Figure shows an improvement in the computation time of 83%. Additionally, the error is negligible with a moderate bucket size of 100 bins. Note that the error in the KS distance directly translates to the error in stability. Since the objective is to evaluate whether the stability suits the proposed algorithm, Figure 8 shows a good scalability improvement with a negligible error not impacting the stability computation.

## 7 SteerAG: Steered Generation and Analysis

StatAG provides an approximation of the entire AG during each iteration, suitable when the AG is used for monitoring purposes and no analytical query is issued. When security analysts are interested in specific analyses, we introduce the *Steered Attack Graph Generation and Analysis* (SteerAG) methodology, reported in Figure 9. According to the workflow (see Figure 1), the steering process is activated with the presence of partial attack paths (PAP in Figure 9) generated using StatAG and an attack path query for the analysis, issued by a human user or an automatic process.

The first step of the steering approach is the *path features training*, which consists of two activities. First, we label the attack paths generated by StatAG as "relevant" when their attack path features correspond to the query requirements and "not relevant" otherwise. As this operation is linear in the number of attack paths generated up to issuing an analysis query in the worst case, it does not represent a costly operation. When a sufficient number of attack paths are labeled, with this number depending on the ML model used (i.e., ten to fifty for decision trees, around 100 for neural networks), the second activity is the actual training of a binary ML-based classification model, where the attack paths are classified according to their assigned label. The classification is based on the vulnerability features of the paths and not on the characteristics of the whole path. In this way, we aim to discover the latent relations between all the relevant attack paths and their vulnerability characteristics based on the ones available. Extracting these relations allows for steering the AG generator by the discovered intervals of vulnerability features.

In our proposal, we use the decision tree classifier [55] as the ML model for its properties that fit well with the steering approach, as it: (i) can be trained quickly and therefore is suitable for progressive data analysis [49], (ii) produces sufficiently powerful classification models from relatively small inputs, allowing its use early on during the progression [55], and (iii) can be easily translated into steering rules, which represent the features of the relevant data [56].

Once the decision tree has been trained on the vulnerability features, the goal of the *vulnerability features extraction* phase is to retrieve the rules to steer the next StatAG generation from the structure of the trained decision tree. We name them *steering rules* and must consider the values of the vulnerability features that generated only the "relevant" paths class. To do so, we consider that the decision tree is built by using the vulnerability features as split points, and a leaf of the tree identifies whether or not the vulnerability

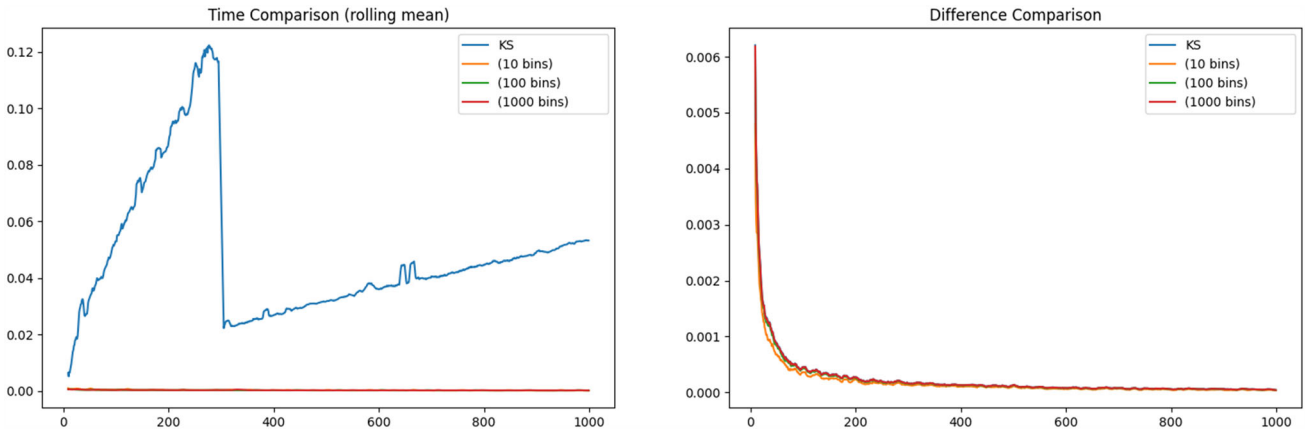
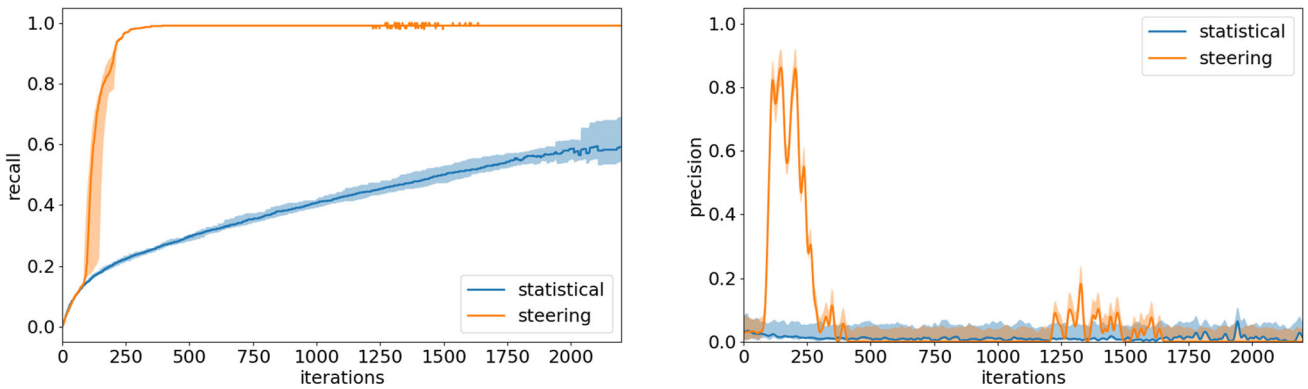



**Fig. 9** SteerAG workflow




(a) Recall. (b) Precision.

**Fig. 10** Comparison of (a) recall and (b) precision for StatAG and SteerAG. Median values are with full-color hue, and alpha-blended areas indicate quartiles

features from the root to the leaf generate "relevant" paths. This makes it possible to compute the inverse mapping from the "relevant" leaves to the root, keeping track of the vulnerability features values that only they have in common. We can build the set of steering rules considering that each path from the root to a "relevant" leaf node represents a *conjunction* of decision rules that must be satisfied for the decision tree to classify it as relevant. Consequently, the logical *disjunction* of all such decision rules generates the set of steering rules.

In the next step of the approach, we use the steering rules to consider only the vulnerabilities that match them before sampling the graph again with random walks, as the StatAG generator mandates. In this way, the newly generated attack paths have a high probability of matching the "relevant" class for the attack path query given in input, thus accelerating the progression of the AG generation towards attack paths answering the query.

Let us note that while the system is running, it continues to use the latest retrieved steering rules until there is a slowdown in the precision of the generation process, indicating that the relevant attack paths have been exhausted. To check that this effectively means having reached the exact answer to the query, another training is launched to adjust the steering with new rules that take into account the cumulative collection of attack paths, effectively repeating the described process until it converges to the exact result (i.e., a new training does not return any additional attack paths). At each iteration of SteerAG, preliminary attack path analysis can be performed to answer the query until all paths are generated. The process is even faster in convergence, guaranteeing analysis results that are identical in quality to the ones conducted on the fully generated AG.

### 7.1 SteerAG Validation

For the validation of SteerAG, we use the setting configurations used for StatAG and described in Section 5.1, with the addition of another experimental parameter that is a set of 100 attack path queries obtained with different combinations of attack path features in the query and varying their ranges

from 0 to 1 with steps of 0.1. This resulted in a total number of 60,000 experiments.

The main evaluation metrics for the steering approach are the *recall* and the *precision*. The former, measured as $\frac{relevant\_paths}{all\_paths}$, informs about the convergence rate of the partial AG to the GT, while the latter, evaluated as $\frac{relevant\_paths}{retrieved\_paths}$, informs about the ability of the decision tree model to correctly retrieve the attack paths that are relevant for the query at every iteration.

Figure 10 reports the recall and precision of the performed experiments. The median values trend for each curve is reported with full-color hue, while the alpha-blended areas identify the variations between the upper and lower quartile values. From the recall trend (Figure 10a) we can observe the very good performance of SteerAG that reaches values very close to 1 after just 250 iterations, with the activation of the steering mechanism starting at iteration 100 on average. In particular, we can recognize two thresholds during SteerAG: between iteration 100 and 150 we observe early partial results (recall lower than 0.5 but fastly increasing), useful for exploration; from iteration 200 until iteration 250 (11% of the execution) the results are mature (recall higher than 0.9), supporting early decision making; finally, from iteration 250 on we observe definitive partial results (recall higher than 0.98).

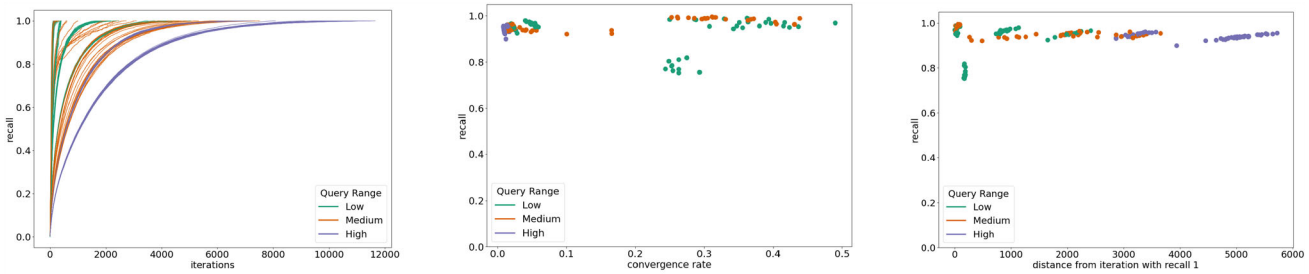
It is interesting to note that without the steering approach, the convergence to the GT is very slow: at the definitive partial results threshold (iteration 250), random sampling has achieved a recall value of just 0.2. Comparing the two trends, we can conclude that the steering approach accelerates the convergence to the GT, saving around 90% of the iterations consistently in all the experiments tested, providing a more timely analysis of attack paths.

While the recall describes the ability of SteerAG to rapidly converge to the GT, the precision informs the analysts about the percentage of attack paths answering the query at each iteration; thus, high values correspond to many paths answering the query. Figure 10b shows the median precision trend during the execution. We can observe that the precision values are consistently high after the activation of the steering mechanism at iteration 100, presenting median values that reach peaks of 0.85. Similarly to the recall trend, precision presents rapid growth in the early iterations, particularly during the generation of early partial results; this shows that the analyst can expect a fast transition from early to mature partial results. At iteration 500 (23% of the execution) it slows down. After 700 iterations (at 55% of the execution), there is another peak of precision up to 0.2 (visible also in the recall trend in Figure 10a. It is due to the retraining of the decision tree to adjust the steering rules after the detection of the precision breakdown. Indeed, a precision breakdown happens only when most of the paths are already retrieved, aligning with low values typical of random sampling (end of steering phase). The fact that the precision has lower median values in the second steering activation than in the first one can be explained by a lower residual probability of finding the few relevant attack paths left (less than 2% at that stage). In contrast, StatAG without the steering mechanism has a constant trend of low precision values because, as expected, the probability of uniformly randomly picking relevant attack paths to the issued query is much lower.

As a final analysis, we are interested in studying how a stricter query affects performance to avoid situations where a very specific query gets stuck due to slow convergence. To validate this, we define three ranges of queries: *low*, when the values of the attack path features are in a range of at most 0.2 (strict query); *medium*, when it is between 0.3 and 0.5; and *high*, when it is over 0.5 (large query). We report different analyses in Figure 11. Figure 11a shows the recall trend of the different experiments for each query range. It highlights three different trends that for the low, medium, and high ranges are respectively the fastest, medium, and slowest convergence to the GT. To quantify the convergence speed to the GT, in Figure 11b we plot the recall values at the point of the maximum convergence of the recall curves. In practice, it corresponds to the peak of the recall curve before its stability. If the maximum convergence rate is low ($< 0.2$) for high recall values ($> 0.8$), then it means that the approach converges slowly to the GT. It is the case of 100% of the experiments with high query ranges, 54% of the experiments with medium ranges, and 25% of the ones with low query ranges. In contrast, the rest of the medium and low-range queries show high convergence speed ($> 0.3$) when the recall is higher than 0.8, indicating quick convergence to the GT. In Figure 11c, we analyze the iterations remaining from peak convergence to retrieve all relevant paths (missing rate). The left part of the plot comprises experiments closer to recall value 1, including all low-range experiments and 70% of medium-range ones. The right one involves cases needing more iterations to reach recall 1, encompassing all high-range experiments and the remaining medium-range ones. Results for low and medium ranges confirm SteerAG's ability to quickly retrieve accurate attack paths even for strict queries. High-range query results seem inferior, but this is due to the large number of attack paths to retrieve, influenced by the sampling size. Precision remains consistently high, reaching convergence rates with recall values exceeding 0.8, even for high-range queries. Overall, we do not observe an effect on the performance.

## 8 Evaluation

After discussing the quantitative validation of the progressive workflow for AG generation –including both StatAG and SteerAG– we show the capabilities of the approach to be




(a) Recall trend. (b) Convergence speed. (c) Missing rate.

**Fig. 11** Analysis of steering mechanism for different query ranges

applied in a real setting and for a representative set of attack path analyses collected from the literature.

## 8.1 Application to large real networks

The first evaluation we provide deals with scalability. In Table 1 we report the works of the literature that address the generation and analysis of attack paths, highlighting the considered maximum number of hosts, vulnerability per host, and whether they generate all paths. In this last case, the approaches define heuristics to prioritize the paths under analysis and avoid the full generation (more details in Section 3). From Table 1 we can observe that the largest experimental settings include either many hosts (50) and few vulnerabilities (5), or a medium number of hosts (15) and vulnerabilities (12 vulnerabilities per host). For the validation of our approach, we already considered networks bigger than those, but here we report an even bigger and real network. We emulated the network of a university department composed of around 250 devices arranged in 5 LANs connected through routers. The network has been scanned for vulnerabilities using commercially available vulnerability scanners (i.e., Tenable Nessus).

Concerning the attack path query, and for the sake of example, we consider the most common risk analysis that requires attack paths with likelihood and impact higher than 0.9, as the most risky situation. We prove later the support of a wide range of analysis queries. Let us note that the generation of the complete AG is intractable for such a size with the classic full generation, but we show that the progressive generation allows having controllable partial results. The stability trend of Figure 12 shows that until iteration 100 the results are early partial, with low and fluctuating stability indicating a still high approximation of the complete AG. The analyst can start exploring in this phase, even by iteration 1, and eventually query the system for her information needs. From iteration 100, high stability values start both for likelihood and impact, thus hypothesizing that it is the time of significant partial results, and supporting tasks like hypothesis formulation. In a real scenario, the first peak of

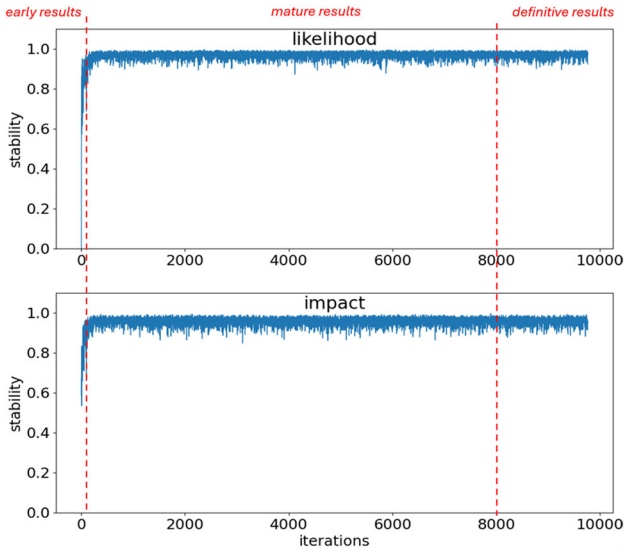



**Fig. 12** AP features stability trend

stability is not enough because it may not be the most accurate scenario because it may slow down due to stability assessment, as happens at iteration 150 of the likelihood trend. However, it is reasonable to observe a constant trend of the stability values before considering the results statistically significant: a reasonable time to consider mature partial results is at iteration 500, where both likelihood and impact show a regular median stability of 0.85, progressively increasing. This phase supports hypothesis testing, comparative or confirmatory analyses, and early decision-making. From this time, the attack path analysis can start, considering that it approximates at least 85% of the complete AG, until reaching very stable results (definitive partial) at iteration 8000.

Meanwhile, the analyst observes the precision trend of Figure 13. She notices a rapid growth at iteration 100, indicating that the SteerAG has been activated, thus indicating the starting point of mature partial results. The precision has an average value of 0.8 until around iteration 8000. When the precision is stable for a longer period (i.e., around iteration 2000), she can consider that most of the attack

**Table 1** SoA settings

| | num. hosts | num. vulns | all paths |
|---|---|---|---|
| Wang et al. (2016) [37] | 7 | 2 | ✓ |
| Tian et al. (2017) [57] | 4 | ND | ✓ |
| George et al. (2018) [58] | 50 | ND | × |
| Yichao et al. (2019) [31] | 50 | 5 | × |
| Li et al. (2019) [25] | 18 | 7 | ✓ |
| Li et al. (2020) [26] | 15 | 12 | ✓ |
| Sun et al. (2022) [18] | 8 | 4 | × |
| Proposed approach | > 250 | > 50 | ✓ |

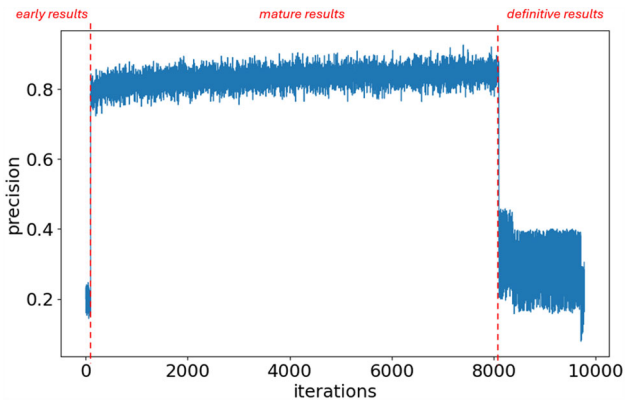



**Fig. 13** Precision trend SteerAG

paths are included in the query and, from this moment on, assumes that the analysis reached definitive partial results (i.e., other incoming paths do not significantly change the attack path feature distributions). When the precision breaks down around iteration 8000, the results are very close to the exact ones (definitive partial results). These observations agree with the stability trend (Figure 12). To give an idea of the advantages provided by this workflow in terms of time, an analyst uses the progressive generation to perform early decision-making in interactive times. Each iteration has an average duration between 0.5 and 5 seconds, and after a few minutes ($\approx 5$) from the beginning, she can start the analysis with mature partial results for any issued query. Additionally, the workflow has the advantage of progressive results feeding. In fact, it may be adapted to automatic processes or agents, where the need for interactive response time is less constrained, but instead results accuracy thresholds gain more importance: the attack path analysis reaches definitive results after a few hours ($\approx$ 12) making it computable in contrast to classic approaches that do not support it.

## 8.2 Coverage of Attack Path Analyses

To conclude the case study, we analyze how the proposed approach can analyze a selected set of queries with the configuration settings used for the validation (Section 5.1). The attack path queries are extracted from state-of-the-art works addressing the systematization of attack path analysis [59, 60]:

**Q1**: Evaluate the risk on the shortest attack paths [61];
**Q2**: Retrieve the attack paths with maximum impact (i.e., impact = 1) [62];
**Q3**: Retrieve the attack paths with maximum likelihood (i.e., likelihood = 1) [63]
**Q4**: Retrieve the attack paths with maximum risk (i.e., likelihood · impact = 1) [4];

noindent **Q5**: Retrieve the attack paths corresponding to *black swan* attacks, i.e., those ones with high impacts (impact > 0.9), but unlikely to happen (likelihood < 0.3) [64];
**Q6**: Retrieve the attack paths corresponding to *gray swan* attacks, i.e., those with very low impacts (impact < 0.3), but highly probable (likelihood > 0.9) [65];
**Q7**: Prioritize the attack paths by risk [6]. Let us note that it corresponds to an ensemble of queries, retrieving first the paths with risk 1, then between 0.9 and 1, and so on. To avoid the complete enumeration of the paths, we stop the retrieval at risk 0.5. Figure 14 reports the recall median trend for the analysis of these queries, where each experiment is repeated 100 times. The median values trend for each curve is reported with full-color hue, while the alpha blended areas identify the variations between the upper and lower quartile values.

Figure 14 illustrates the rapid convergence of recall for the majority of queries (Q1-Q5 and Q7) to the ground truth (GT), thereby demonstrating the substantial utility of the proposed approach in facilitating attack path analysis. Queries Q2, Q3, and Q4 specifically target precise values of attack path features, whereas Q1, Q5, and Q7 involve more intricate requests necessitating multiple traversals within the same Attack Graph (AG). Conversely, query Q6, pertaining to gray swan attacks, exhibits the lowest performance, requiring a median of 3500 iterations for complete path retrieval. This diminished performance is attributable to the increased number of required traversals and the multiplicity of paths

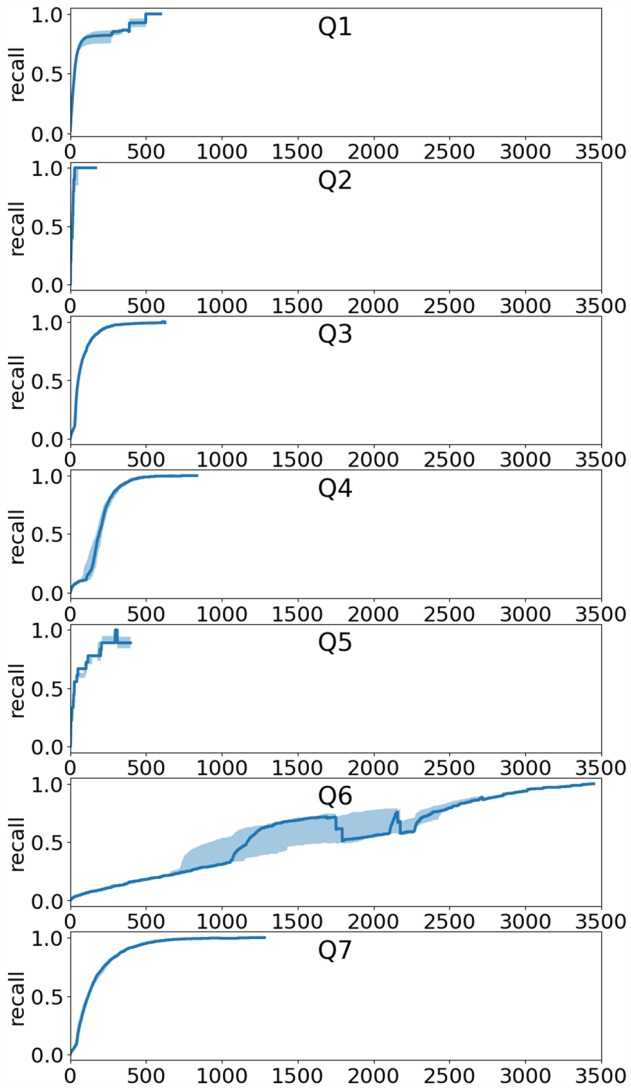



**Fig. 14** Recall for different queries

satisfying the query. Despite the slower convergence, the approach still yields approximate answers for Q6.

This case study effectively demonstrates the capability of the proposed approach to conduct attack path analysis aligning with state-of-the-art attack path queries. Furthermore, it is important to note that queries can be initiated at any point during execution, with the approach automatically activating its steering mechanism upon query reception.

## 9 Conclusions

This paper presented StatAG and SteerAG as core components of a new workflow to generate and analyze AGs based on progressive data analysis. The former generates statistically significant partial AGs, and the latter accelerates AG generation based on attack path queries. Our approach adapts better to network changes without recomputing the entire AG, a feature lacking in traditional AG generation methods. This is especially guaranteed by the improved performance of the weighted path sampling and revised KS algorithms that ensure scalability in large networks without lowering the accuracy of the results. Extensive experiments and a case study validate the capability to analyze large networks and efficiently support common attack path analyses. All the materials and the source code of all components are publicly available at https://github.com/XAIber-lab/ProgressiveAttackGraph.

A first limitation of the proposed approach is the application of StatAG and SteerAG to *topological* AGs. We believe they can be adjusted for attack path analysis over logical ones, for example, adapting the path construction with model-checking [66], which we consider for future work. Another aspect to expand on is the consideration of likelihood and impact as attack path features since they are the main ones used in the literature on cyber risk management. Still, others could be considered, dependent on the application domain (e.g., cloud [67] and automotive [68]), that we will explore in future works. Finally, we also suggest exploring other ML models, e.g., Reinforcement Learning [69] and Graph Neural Networks [70] that, while allowing to capture non-linear relations, are challenging to adapt to our approach due to their resource-intensive training and their black-box nature that makes it difficult to retrieve the steering rules.

Future works may study automated ways to set progressive thresholds based on real-time indicators. Another opportunity involves the improvement of the decision tree predictions after the precision breakdown. Currently, we re-train the decision tree with the newly incoming paths, but more advanced strategies to optimize the re-training of the ML model may be adopted. Finally, according to the newly emerging application of mixed initiative involving humans and AI, the adaptation of this workflow to autonomous systems, agentic systems where multiple queries are dedicated to different agents with a Human-on-the-loop supervising the process, and Human-in-the-loop, where humans and AI agents provide a similar level of agency, all represents interesting future research avenues.

**Acknowledgements** This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU and by FaRADAI Project funded by European Commission under European Defence Fund Grant 101103386. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union nor the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

**Author Contributions** AP: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization CC: Conceptualization, Validation, Formal analysis, Investigation, Data Curation,

Visualization MA: Conceptualization, Methodology, Validation, Formal analysis, Investigation, Writing - Original Draft, Writing - Review & Editing, Supervision.

**Data Availability** Data is open source and provided within the manuscript.

## Declarations

**Competing interests** The authors declare no competing interests.



## References

1. Dambra, S., Bilge, L., Balzarotti, D.: Sok: Cyber insurance–technical challenges and a system security roadmap, in *2020 IEEE Symposium on Security and Privacy (SP)* (IEEE), pp. 1367–1383 (2020)
2. Gonzalez-Granadillo, G., Dubus, S., Motzek, A., Garcia-Alfaro, J., Alvarez, E., Merialdo, M., Papillon, S., Debar, H.: Dynamic risk management response system to handle cyber threats. Futur. Gener. Comput. Syst. **83**, 535 (2018)
3. Zengy, J., Wang, X., Liu, J., Chen, Y., Liang, Z., Chua, T.S., Chua, Z.L.: Shadewatcher: Recommendation-guided cyber threat analysis using system audit records, in *2022 IEEE Symposium on Security and Privacy (SP)* (IEEE), pp. 489–506 (2022)
4. Woods, D.W., Böhme, R.: Sok: Quantifying cyber risk, in *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 211–228 (2021)
5. Kaynar, K.: A taxonomy for attack graph generation and usage in network security. Journal of Information Security and Applications **29**, 27 (2016)
6. Zenitani, K.: Attack graph analysis: An explanatory guide. Computers & Security **126**, 103081 (2023)
7. Stergiopoulos, G., Dedousis, P., Gritzalis, D.: Automatic analysis of attack graphs for risk mitigation and prioritization on large-scale and complex networks in industry 4.0. Int. J. Inf. Secur. **21**(1), 37 (2022)
8. Li, Z., Zeng, J., Chen, Y., Liang, Z.: Attackg: Constructing technique knowledge graph from cyber threat intelligence reports, in *Computer Security – ESORICS 2022* (Springer International Publishing), pp. 589–609 (2022)
9. Zenitani, K.: A scalable algorithm for network reachability analysis with cyclic attack graphs, Journal of Computer Security pp. 1–27 (2022)
10. Palma, A., Bonomi, S.: A workflow for distributed and resilient attack graph generation, in *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)* (IEEE), pp. 185–187 (2023)
11. Angelini, M., Santucci, G., Schumann, H., Schulz, H.J.: A review and characterization of progressive visual analytics. Informatics **5**, 31 (2018)
12. Fekete, J.D., Primet, R.: Progressive analytics: A computation paradigm for exploratory data analysis, arXiv preprint arXiv:1607.05162 (2016)
13. A. Palma, M. Angelini, It is Time To Steer: A Scalable Framework for Analysis-Driven Attack Graph Generation, in *Computer Security ESORICS 2024*, ed. by J. Garcia-Alfaro, R. Kozik, M. Choras, S. Katsikas (Springer Nature Switzerland, Cham, 2024), pp. 229–250. https://doi.org/10.1007/978-3-031-70903-6_12
14. Ingols, K., Lippmann, R., Piwowarski, K.: Practical Attack Graph Generation for Network Defense, in *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, 121–130 (2006)
15. Security, T.N.: Nessus. tool, Tenable Network Security (2022). www.tenable.com/products/nessus
16. Jajodia, S., Noel, S.: Topological vulnerability analysis, in *Cyber situational awareness: Issues and research* (Springer), 139–154 (2009)
17. Feng, Y., Wang, L., Zhang, J., Cai, Z., Gan, Y.: Generation Method of Network Attack Graph Based On Greedy Heuristic Algorithm. International Journal of Hybrid Information Technology **10**(6), 23 (2017)
18. Sun, W., Li, Q., Wang, P., Hou, J.: Heuristic Network Security Risk Assessment Based on Attack Graph, in Cloud Computing, pp. 181–194. Springer International Publishing, Cham (2022)
19. Yuan, B., Pan, Z., Shi, F., Li, Z.: An Attack Path Generation Methods Based on Graph Database. 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) **1**, 1905–1910 (2020)
20. Ramos, A., Lazar, M., Holanda Filho, R., Rodrigues, J.J.: Model-based quantitative network security metrics: A survey. IEEE Communications Surveys & Tutorials **19**(4), 2704 (2017)
21. Angelini, M., Bonomi, S., Borzi, E., Pozzo, A.D., Lenti, S., Santucci, G.: An attack graph-based on-line multi-step attack detector, in *Proceedings of the 19th International Conference on Distributed Computing and Networking* (Association for Computing Machinery, New York, NY, USA), ICDCN '18. https://doi.org/10.1145/3154273.3154311 (2018)
22. Kaynar, K., Sivrikaya, F.: Distributed Attack Graph Generation. IEEE Trans. Dependable Secure Comput. **13**(5), 519 (2016)
23. Sabur, A., Chowdhary, A., Huang, D., Alshamrani, A.: Toward scalable graph-based security analysis for cloud networks. Comput. Netw. **206**, 108795 (2022)
24. Chen, Y., Liu, Z., Liu, Y., Dong, C.: Distributed Attack Modeling Approach Based on Process Mining and Graph Segmentation. Entropy **22**(9), 1026 (2020). https://doi.org/10.3390/e22091026
25. Li, M., Hawrylak, P., Hale, J.: Concurrency Strategies for Attack Graph Generation, in *2019 2nd International Conference on Data Intelligence and Security (ICDIS)*, 174–179 (2019)
26. Li, M., Hawrylak, P.J., Hale, J.: Implementing an Attack Graph Generator in CUDA, in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 730–738 (2020)
27. Cook, K., Shaw, T., Hawrylak, P., Hale, J.: Scalable attack graph generation, in *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, pp. 1–4 (2016)
28. Cao, N., Lv, K., Hu, C.: An Attack Graph Generation Method Based on Parallel Computing, (Springer International Publishing. Cham **11287**, 34–48 (2018). https://doi.org/10.1007/978-3-030-03026-1_3
29. Li, T., Jiang, Y., Lin, C., Obaidat, M., Shen, Y., Ma, J.: DeepAG: Attack Graph Construction and Threats Prediction with Bi-directional Deep Learning, IEEE Transactions on Dependable and Secure Computing pp. 1–1 (2022)

30. Grover, A., Leskovec, J.: node2vec: Scalable Feature Learning for Networks, in *Proceedings of the 22nd ACM SIGKDD* (Association for Computing Machinery, New York, NY, USA), KDD '16, pp. 855–864 (2016)
31. Yichao, Z., Tianyang, Z., Xiaoyue, G., Qingxian, W.: An Improved Attack Path Discovery Algorithm Through Compact Graph Planning. IEEE Access **7**, 59346 (2019)
32. Mao, B., Liu, J., Lai, Y., Sun, M.: Mif: A multi-step attack scenario reconstruction and attack chains extraction method based on multi-information fusion. Comput. Netw. **198**, 108340 (2021)
33. Zhao, J., Tang, T., Bu, B., Li, Q.: Graph neural network-based attack prediction for communication-based train control systems, CAAI Transactions on Intelligence Technology (2024)
34. Presekal, A., Stefanov, A., Rajkumar, V.S., Palensky, P.: Attack graph model for cyber-physical power systems using hybrid deep learning. IEEE Trans. Smart Grid **14**(5), 4007 (2023)
35. Gonda, T., Pascal, T., Puzis, R., Shani, G., Shapira, B.: Analysis of attack graph representations for ranking vulnerability fixes., in *GCAI*, pp. 215–228 (2018)
36. Nichols, W., Hawrylak, P., Hale, J., Papa, M.: Introducing priority into hybrid attack graphs, in *Proceedings of the 12th Annual Conference on Cyber and Information Security Research* (Association for Computing Machinery, New York, NY, USA), CISRC '17, pp. 1–4. https://doi.org/10.1145/3064814.3064826 (2017)
37. Wang, S., Tang, G., Kou, G., Chao, Y.: An attack graph generation method based on heuristic searching strategy, in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pp. 1180–1185 (2016)
38. Salayma, M., Lupu, E.C.: Threat Modelling in Internet of Things (IoT) Environment Using Dynamic Attack Graphs (2023). ArXiv:2310.01689 [cs]
39. Guia, J., Soares, V.G., Bernardino, J.: Graph databases: Neo4j analysis. ICEIS **1**, 351–356 (2017)
40. Zhang, D., Qian, K., Zhang, P., Mao, S., Wu, H.: Alert correlation analysis based on attack path graph, in *2017 IEEE Conference on Energy Internet and Energy System Integration (EI2)*, pp. 1–6 (2017)
41. GhasemiGol, M., Ghaemi-Bafghi, A., Takabi, H.: A comprehensive approach for network attack forecasting. Computers & Security **58**, 83 (2016)
42. Alhaj, T.A., Siraj, M.M., Zainal, A., Idris, I., Nazir, A., Elhaj, F., Darwish, T.: An effective attack scenario construction model based on identification of attack steps and stages, International Journal of Information Security pp. 1–16 (2023)
43. Ahmadian Ramaki, A., Rasoolzadegan, A.: Causal knowledge analysis for detecting and modeling multi-step attacks. Security and Communication Networks **9**(18), 6042 (2016)
44. Wang, C.H., Chiou, Y.C.: Alert correlation system with automatic extraction of attack strategies by using dynamic feature weights. International Journal of Computer and Communication Engineering **5**(1), 1 (2016)
45. Wang, Y., Guo, Y., Fang, C.: An end-to-end method for advanced persistent threats reconstruction in large-scale networks based on alert and log correlation. Journal of Information Security and Applications **71**, 103373 (2022)
46. Saad, S., Traore, I.: Extracting attack scenarios using intrusion semantics, in *Foundations and Practice of Security: 5th International Symposium, FPS 2012, Montreal, QC, Canada, October 25-26, 2012, Revised Selected Papers 5* (Springer), pp. 278–292 (2013)
47. Nadeem, A., Verwer, S., Moskal, S., Yang, S.J.: Alert-Driven Attack Graph Generation Using S-PDFA. IEEE Trans. Dependable Secure Comput. **19**(2), 731 (2022)
48. Hassan, W.U., Bates, A., Marino, D.: Tactical provenance analysis for endpoint detection and response systems, in *2020 IEEE Symposium on Security and Privacy (SP)* (IEEE), pp. 1172–1189 (2020)
49. Hogräfer, M., Angelini, M., Santucci, G., Schulz, H.J.: Steering-by-example for Progressive Visual Analytics. ACM Transactions on Intelligent Systems and Technology **13**(6), 1–96 (2022)
50. Lee, C.Y.: An algorithm for path connections and its applications. IRE Trans. Electron. Comput. **3**, 346 (1961)
51. Li, R.H., Yu, J.X., Qin, L., Mao, R., Jin, T.: On random walk based graph sampling, in *2015 IEEE 31st international conference on data engineering* (IEEE), pp. 927–938 (2015)
52. Sproull, N.L.: *Handbook of research methods: A guide for practitioners and students in the social sciences* (Scarecrow press) (2002)
53. Massey, F.J.: The kolmogorov-smirnov test for goodness of fit. J. Am. Stat. Assoc. **46**(253), 68 (1951)
54. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M.: Édouard Duchesnay, Scikit-learn: Machine learning in python. J. Mach. Learn. Res. **12**(85), 2825 (2011)
55. Kotsiantis, S.B.: Decision trees: a recent overview. Artif. Intell. Rev. **39**, 261 (2013)
56. Dimitriadou, K., Papaemmanouil, O., Diao, Y.: Explore-by-example: An automatic query steering framework for interactive data exploration, in *Proceedings of the international conference on Management of data*, pp. 517–528 (2014)
57. Tian, J.W., Li, X., Tian, Z., Qi, W.H.: Network attack path reconstruction based on similarity computation, in *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery* (IEEE, Guilin, 2017), pp. 2457–2461
58. George, G., Thampi, S.M.: A Graph-Based Security Framework for Securing Industrial IoT Networks From Vulnerability Exploitations. IEEE Access **6**, 43586 (2018)
59. Landoll, D.J.: *Information Security Policies, Procedures, and Standards: A Practitioner's Reference* (CRC Press) (2017)
60. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.: Automated generation and analysis of attack graphs, in *Proceedings 2002 IEEE Symposium on Security and Privacy*, pp. 273–284 (2002)
61. Noel, S., Jajodia, S.: Metrics suite for network attack graph analytics, in *Proceedings of the 9th Annual Cyber and Information Security Research Conference*, pp. 5–8 (2014)
62. Kotenko, I., Doynikova, E.: Security Assessment of Computer Networks Based on Attack Graphs and Security Events, in Information and Communication Technology, vol. 8407, pp. 462–471. Springer, Berlin Heidelberg (2014)
63. Kavallieratos, G., Katsikas, S.: Attack Path Analysis for Cyber Physical Systems, in Computer Security, pp. 19–33. Springer International Publishing, Cham (2020)
64. Aven, T.: On the meaning of a black swan in a risk context. Saf. Sci. **57**, 44 (2013)
65. Khakzad, N., Khan, F., Amyotte, P.: Major accidents (gray swans) likelihood modeling using accident precursors and approximate reasoning. Risk Anal. **35**(7), 1336 (2015)
66. Clarke, E.M.: Model checking, in *Foundations of Software Technology and Theoretical Computer Science* (Springer), pp. 54–56 (1997)
67. Pauley, E., Sheatsley, R., Hoak, B., Burke, Q., Beugin, Y., McDaniel, P.: Measuring and mitigating the risk of ip reuse on public clouds, in *2022 IEEE Symposium on Security and Privacy (SP)* (IEEE), pp. 558–575 (2022)

68. Macher, G., Armengaud, E., Brenner, E., Kreiner, C.: A review of threat analysis and risk assessment methods in the automotive context, in *35th International Conference, SAFECOMP 2016* (Springer), pp. 130–141 (2016)
69. Terranova, F., Lahmadi, A., Chrisment, I.: Scalable and generalizable rl agents for attack path discovery via continuous invariant spaces, in,: 28th International Symposium on Research in Attacks. Intrusions and Defenses (RAID) **2025**, 18 (2025)
70. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems **32**(1), 4 (2021)

# Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com